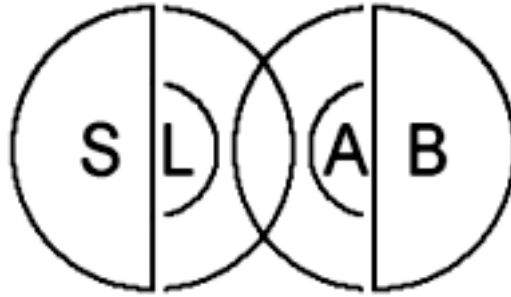


# SLAB User Manual



**Joel D. Miller**

SLAB is a software-based real-time virtual acoustic environment (VAE) rendering system being developed in the Spatial Auditory Displays Lab at NASA Ames Research Center.

Note: The SLAB User Release is a work-in-progress. Functionality described herein is subject to change. This document will be revised frequently.

## Contents

- [Introduction](#)
  - [System Requirements](#)
  - [Installation](#)
  - [Citing SLAB](#)
  - [Acknowledgements](#)
- [Overviews](#)
  - [SLAB Render API](#)
  - [Coordinate System](#)
  - [Sound Sources](#)
  - [HRTF Databases](#)
  - [Error Handling](#)
  - [Tips](#)
- [SLAB Render Plug-Ins](#)
  - [Overview](#)
  - [Starter Project](#)
- [Source Code](#)
  - [SLABClient](#)

- [rdiotic](#)
- [rplugin](#)
- [SLABX](#)
- [Installer](#)
- [Applications and Utilities](#)
  - [SLABscape](#)
  - [SLABServer](#)
  - [slabtools](#)
  - [Example Application](#)
- [Appendix](#)
  - [Glossary](#)
  - [Known Issues](#)

**See also:** [SRAPI Reference Manual](#)

SLAB User Manual v5.3.0

Copyright 2001-2003 U.S. Government as represented by the  
Administrator of the National Aeronautics and Space Administration.

Initiated: July 5, 2000

Last Updated: September 10, 2003

# Introduction

- [System Requirements](#)
  - [Installation](#)
  - [Citing SLAB](#)
  - [Acknowledgements](#)
- 

## System Requirements

The software and hardware components listed below are required and/or recommended for developing SLAB applications.

### Software

*Operating System:* Windows2000

SLAB is being developed under Windows2000. In theory, SLAB should be compatible with Windows98SE/ME, but this has not been tested. Client applications have run successfully under WindowsNT 4.0, Service Pack 3. NT host-mode applications, however, are likely to fail due to NT's DirectSound emulation.

*Low-latency Audio Interface:* DirectSound 8.1

SLAB uses DirectSound to achieve low-latency audio output (e.g., ~24ms). The sound card used with SLAB must have a non-emulated DirectSound driver installed. To run the SLABscape and SLABserver applications, the DirectSound Runtime should be installed. To develop SLAB applications, the DirectX SDK should be installed. If you have a previous DirectSound version installed, it might not be necessary to upgrade. First, give SLAB a try; if you encounter problems, you might want to upgrade. DirectSound 6 Runtime is installed with Windows98SE. The Microsoft DirectX download site is: <http://www.microsoft.com/directx/>. Developer information is available at: <http://msdn.microsoft.com/directx/>.

*Note regarding latency under Windows2000 vs Windows98SE:*

Under Windows2000, the DirectSound driver accesses the hardware through a WDM driver. Some WDM drivers use the Windows K Mixer which adds approximately 20ms of latency to sound output. Windows98SE WDM drivers also appear to have this additional latency (versus VXD drivers). SLAB's total internal latency is approximately equal to the sum of the DirectSound buffer size, the smooth-time value, and the K Mixer latency (for WDM drivers).

*Compiler (optional):* Microsoft Visual C++ .NET

A compiler is necessary for SLAB application development. SLAB is being developed using Microsoft Visual C++ .NET. The SLAB header files and libraries have not been tested under other development environments. Missing library errors have been noticed when linking with the SLAB libraries using Microsoft Visual C++ 6.0. Thus, the SLAB libraries may not be compatible with pre-.NET versions of Microsoft Visual C++.

### Hardware

#### SLAB Server or Host Workstation

*Computer Workstation:*

Minimum recommended requirements: Intel 650MHz PIII CPU, 256MB RAM.

*DirectSound Sound Card:*

Windows98SE: Turtle Beach Montego II Sound Card (Montego II digital output is not supported under Windows2000)

Windows2000: Creative Audigy

SLAB Client/Server Network

*100MBit Ethernet Card*

*100MBit Ethernet Hub*

*2 100BaseTX (RJ-45) Ethernet Cables (client to hub and hub to SLAB Server)*

Digital Output (optional)

*Digital Output Device:*

Windows98SE: Turtle Beach Montego II Digital I/O Upgrade

Windows2000: Creative Audigy

*Digital-to-Analog Converter:*

Lucid Technology ADA1000 A/D, D/A Converter

Headphone Display

*Headphone Amplifier (optional):*

Symetrix SX204 Headphone Amplifier

*Headphones:*

Sennheiser HD 545 Reference Headphones

[Top](#)

---

## **Installation**

SLAB User Release Installation Instructions

SLAB is available for download at the SLAB Home Page:

<http://human-factors.arc.nasa.gov/SLAB>.

For installation instructions, see the *SLAB User Release Installation Instructions* under the *Downloads* section of the SLAB Home Page.

Windows Sound Schemes

When listening to SLAB with headphones it is probably best to disable system sounds by selecting the "No Sounds" sound scheme under Control Panel | Sounds and Multimedia | Sounds | Scheme. Otherwise, an uncomfortably loud system sound might occur while listening to

SLAB.

### Speakers Property

Some sound peripherals process the output signal based on the type of display attached to the output (e.g., headphones versus desktop speakers). When using SLAB, it is best to select an unprocessed output path under Control Panel | Sounds and Multimedia | Audio | Advanced | Speakers | Speaker Setup.

[Top](#)

---

## Citing SLAB

As a courtesy, the use of SLAB in published research should be acknowledged in the publication by citing the SLAB Home Page:

[1] <http://human-factors.arc.nasa.gov/SLAB>

The preferred paper reference describing the SLAB User Release and its implementation:

[2] Miller, J. D. and Wenzel, E. M., "Recent Developments in SLAB: A Software-Based System for Interactive Spatial Sound Synthesis," Proceedings of the International Conference on Auditory Display, ICAD 2002, Kyoto, Japan, pp. 403-408, 2002.

[Top](#)

---

## Acknowledgements

I would like to thank Beth Wenzel for her support, feedback, and for making SLAB possible, Jonathan Abel for providing numerous physical modeling and signal processing examples, Mark Anderson for programming assistance, and both Mark Anderson and Durand Begault for testing the waters and offering many helpful suggestions. I would also like to thank Marlene Hernan, Robert Padilla, and Robin Orans.

--joel

## Developers

Joel Miller	lead designer and programmer
Jonathan Abel	physical modeling and signal processing MATLAB scripts
Mark Anderson	TrakLib Fastrak driver, SLABscape 3D View, SLABscape SLABScript Editor, sockets and registration database assistance
Mitch Clapp	SLABscape 3D View models and textures, web-based registration code

[Top](#)

---

## [SLAB User Manual](#)

Last Updated: September 5, 2003

# Overviews

- [SLAB Render API](#)
  - [Coordinate System](#)
  - [Sound Sources](#)
  - [HRTF Databases](#)
  - [Error Handling](#)
  - [Tips](#)
- 

## SLAB Render API

When developing SLAB applications, SLAB is accessed through the SLAB Render API (SRAPI) encapsulated in the CSLABAPI interface. CSLABAPI is an abstract base class used to access the CSLABHost and CSLABClient objects. If you wish to run SLAB locally (i.e., on the same machine as the SLAB application), the CSLABHost object should be used. Conversely, if you wish to run SLAB remotely (i.e., using a network and a dedicated SLAB server workstation), the CSLABClient object should be used. By using the CSLABAPI interface, you can select between local or remote usage at run-time. All host, client, and server functions are available through CSLABAPI. If the function isn't supported for the selected network mode, the function does nothing (e.g. if you allocate a host object and call a client function, the function simply returns).

Note: the "Server Functions" seen in SLABAPI.h are part of the CSLABServer class interface. This class is for writing SLAB server applications and is not included in the SLAB User Release. The SLAB User Release contains the *SLABServer* application for providing SLAB server functionality.

Further information on the SLAB Render API is available in the [SRAPI Reference Manual](#).

[Top](#)

---

## Coordinate System

SLAB uses a right-handed, FLT (Front-Left-Top) coordinate system (i.e., if fingers curled from front to left, thumb points to top).

### Location

+x front, through nose  
+y left, through left ear  
+z top, through top of head

### Orientation

-yaw to right  
+yaw to left  
-pitch up

+pitch down  
+roll right  
-roll left

#### Polar

+azimuth to right  
-azimuth to left  
+elevation up  
-elevation down  
+range forward  
-range backward

#### Convolvotron and Polhemus azimuth and elevation

These definitions are compatible with the Polhemus Isotrak and Fastrak head trackers and the Crystal River Engineering Convolvotron coordinate systems with the following exceptions:

SLAB azimuth = - Convolvotron and Polhemus azimuth  
SLAB and Convolvotron elevation = - Polhemus elevation

#### Polhemus transmitter and receiver

The physical placement of the Polhemus transmitter and receiver often results in the following configuration (as specified on the Fastrak transmitter): +X forward, +Y right, +Z down. Since SLAB uses +Z up and +Y left, the signs of Y, Z, Yaw, and Pitch must be changed if your tracker driver follows this convention.

[Top](#)

---

## **Sound Sources**

SLAB supports two types of sound sources, Windows wave files (see SrcFile()) and software signal generation (see the SrcGen functions). All sound sources must be allocated before calling RenderStart() to initiate real-time rendering. A sound sources is given an ID value when it is allocated. This ID is passed to the source control functions. The number of sources available is limited by the computational resources of the SLAB workstation.

To change the source allocation configuration, call SrcFree() to free all sound sources. SrcFree() cannot be called while processing. Once the previous sources have been freed, a new source configuration can be allocated.

To generate sound sources not supported by the SrcGen functions, use SrcFile() in conjunction with a Windows wave file editing application. Since SrcFile() can loop wave files continuously, it can be used to create fixed periodic sound sources.

There are two methods of muting a sound source:

- Call SrcGain() using a dB gain value much lower than -97.0 dB (-96.3 dB is the dynamic range of 16-bit integer). Note: this does not affect the processing performed. The computational load remains the same.
- Call SrcEnable() to disable the source. The source is no longer rendered, eliminating the source entirely. Disabling a source reduces computational load.

**See Also:** CSLABAPI Source Functions in the [SRAPI Reference Manual](#).

[Top](#)

---

## HRTF Databases

SLAB HRTF (Head-Related Transfer Function) databases contain the HRIR (Head-Related Impulse Response) and ITD (Interaural Time Delay) information needed to perform spatialization. The default SLAB HRTF database is "jdm.slh" and the default HRTF directory is "<install dir>\HRTF". Use the SLABscape application to change these defaults (see menu item: SLAB | SLAB Registry Settings...).

### SLAB HRTF Database Format - Version 1

Version 1 databases are binary files with all values stored in 16-bit integer. They typically have the suffix .dat. The following parameters are assumed:

Azimuth, number of = 13, ( 180 150 120 90 60 30 0 -30 -60 -90 -120 -150 -180 )  
 Azimuth increment = 30  
 Azimuth zero index = 6  
 Elevation, number of = 11, ( 90 72 54 36 18 0 -18 -36 -54 -72 -90 )  
 Elevation increment = 18  
 Elevation zero index = 5  
 Number of ears = 2  
 Number of ITD points = 143  
 ITD positive = left ear lag  
 ITD negative = right ear lag  
 Number of HRIR points = 256  
 Number of HRIR sample bytes = 2 (16-bit integer)  
 Maximum HRIR sample value = 32767  
 Minimum HRIR sample value = -32768  
 Float HRIR sample scaling factor = 32768.0f

The table below illustrates the HRIR and ITD data storage format (ordered by individual data values):

AZ	EL	
180,	90, left	ear, hrir pt 0 <i>Left ear points</i>
...		hrir pt 255
180,	90, right	ear, hrir pt 0 <i>Right ear points</i>
...		hrir pt 255
180,	72, left	ear, hrir pt 0 <i>Elevations (grouped by azimuth)</i>
...	-90, right	ear, hrir pt 255
150,	90, left	ear, hrir pt 0 <i>Azimuths</i>
...		
-180,	-90, right	ear, hrir pt 255

```

180,  90, delay
180,  72, delay
...
-180, -90, delay

```

*Delays*

## SLAB HRTF Database Format - Version 2

Version 1 to Version 2 changes:

- 1) The database now contains a header.
- 2) The 16-bit integer data type has been changed to single-precision floating-point.
- 3) The database files typically have a .slh suffix.

### HRTF Database Header

```

typedef struct
{
short  nVersion;
char   strName[32];           // subject's name
char   strDate[8];           // date measured
char   strComment[256];      // text comment
short  nAzInc;               // degree increment of az locations
short  nElInc;               // degree increment of el locations
short  nNumPts;              // number of HRIR points per entry
long   lSampleRate;          // sample rate when capturing HRIR
} HRTFHeader;

```

The HRIR and ITD data immediately follow the header as specified above for Version 1. The azimuth and elevation increments and the number of HRIR points are now determined from the header.

**See Also:** CSLABAPI::LstHRTF() in the [SRAPI Reference Manual](#).

[Top](#)

---

## Error Handling

There are two error catching modes in SLAB: Non-Process Time and Process-Time. When SLAB is processing (i.e., after a call to RenderStart()), SLAB is in the Process-Time error catching mode. When SLAB is not processing (i.e., after the SLAB object is allocated, after a call to RenderStop(), or after an error), SLAB is in the Non-Process Time error catching mode.

Why two modes? Two reasons:

1. API Thread and Process Thread

SLAB runs in two threads of execution, the API Thread (same as the user's thread) and the Process Thread, a thread created during a call to RenderStart(). Errors can occur inside the Process Thread without an API call (e.g., a state error in a Render plug-in not directly connected to an API call). The user cannot check the return value of an API function to catch this error (unless "polling" and then only during the next "error poll", see below).

## 2. Update Loop in Client/Server Mode

Typically, after a call to `RenderStart()`, the SLAB user will be updating an acoustic scene in some form of update loop (e.g. capturing listener position information with a head tracker or generating a source trajectory). For an update rate of 120Hz, the update loop is executed every 8.3ms. In client/server mode, it would be inefficient to send error packets from the server to the client for every SRAPI call in the update loop.

## **Blocking and Non-Blocking Functions**

In client/server mode, the SRAPI functions are broken into two groups, blocking functions and non-blocking functions. Blocking functions wait for an error packet to be returned from the server, non-blocking functions do not. Blocking functions are typically called prior to processing. Most non-blocking functions can be called at any time. Whether a function is blocking or non-blocking is documented in the function's Remarks section in the [SRAPI Reference Manual](#).

## **Non-Process-Time Error Catching**

### Function Return Values

Almost all SRAPI calls return `SLABError`. Non-Process-Time errors are typically caught with function return values. Non-blocking function errors in client/server mode, however, can only be caught by polling a blocking function (polling is discussed below).

## **Process-Time Error Catching**

There are two methods for catching Process-Time errors:

### 1. Error Messaging

When using Error Messaging, SLAB sends error messages to the user's application via Windows messaging. Before calling `RenderStart()`, call `SetNotify()` specifying a notification window and message ID value. If a Process-Time error occurs, the notification message will be sent to the notification window. To catch the message, map the notification message ID to a message handler using a `CWnd` message map. Use an error query function to verify the notification corresponds to an error (notifications can also be used to indicate the completion of one-shot playback).

### 2. Error Polling

Whenever an error occurs, SLAB enters an "error state." In an error state, all SRAPI functions return the current error. Error Polling refers to catching *existing* errors via function return values or error status functions. In other words, the user is looking for an error not necessarily caused by the function itself. For example, consider several successive non-blocking SRAPI function calls in client/server mode. If the last function call returns an error, the error actually belongs to a previous function call. This delay is due to the time it takes the error to propagate through the messaging system and the network

## **Error Catching Summary**

	<u>Host Mode Methods</u>	<u>Client/Server Mode Function Types</u>	<u>Client/Server Mode Methods</u>
Non-Process Time	return value, poll	Blocking Function	return value, poll
		Non-Blocking Function	blocking function poll
		Blocking Function	return value, poll, message

Process Time	return value, poll, message	Non-Blocking Function, Process Thread	poll, message
--------------	-----------------------------	---------------------------------------	---------------

## **Error State**

SLAB enters an "error state" whenever an error occurs.

In an error state:

- rendering is stopped
- the sound output stream is stopped
- script processing is stopped
- all functions returning SLABError return the current error
- error information can be queried via ErrorState(), Error(), ErrorString(), and ErrorStack()
- error state maintained until explicitly cleared with ErrorClear()

## **Error Functions Overview**

The following functions exist for querying error information and clearing error state:

- ErrorState(): returns true if in error state, false if not
- Error(): returns the current SLABError
- ErrorString(): returns a string describing the current error
- ErrorStack(): returns a string providing call stack error information for the current error
- ErrorClear(): clears error state
- SetNotify(): sets the notification window and the notification message ID value
- LogName(): sets the name of the log file used for logging error information in host-mode
- Reset(): resets SLAB

**See Also:** CSLABAPI Error Functions in the [SRAPI Reference Manual](#).

[Top](#)

---

## **Tips**

Tips for writing psychoacoustic experimentation software, etc.:

- To disable source-listener range-dependent gain scaling use SrcSpread( 0 ).
- To place sound sources by specifying azimuth and elevation use LstPosition( 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) and SrcLocatePolar( idSrc, az, el, range ). Note: SLAB does not have a listener-relative source location function.
- To specify sound source time delays use LstPosition( 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) and SrcLocatePolar( idSrc, az, el, dSoundSpeed \* time\_delay ).

[Top](#)

---

[SLAB User Manual](#)

Last updated: May 29, 2003

# SLAB Render Plug-Ins

- [Overview](#)
- [Starter Project](#)

## Overview

All rendering in SLAB is performed with SLAB Render plug-ins. SLAB Render plug-ins are subclassed from CRPlugIn, built as DLLs, and placed in the SLAB bin\rplugin directory. All plug-ins found in this directory will be read into memory when a CSLABAPI object is allocated. The SRAPI Render Functions exist for querying and selecting plug-ins. Through the CScene class, Render plug-ins are given access to SLAB's scene parameters, input delay lines, and output stream.

**See Also:** CSLABAPI Render Functions, CRPlugIn, and CScene in the [SRAPI Reference Manual](#).

[Top](#)

## Starter Project

The *rplugin* and *rdiotic* projects in the SLAB src\ directory can be used as Render plug-in examples. The *rplugin* project is also used as a starter project for creating user plug-ins. The *rdiotic* project contains the source code for the SLAB monotonic and diotic renderers and shows how to place multiple plug-ins in one DLL.

Follow the steps below to create your own plug-in:

1. In MSVC, create a new DLL project using **New | Projects | Win32 Dynamic-Link Library**. Choose "A simple DLL project." when prompted for the type of DLL project to create.
2. Delete the generated .cpp from the project and from the directory.
3. Copy rplugin.cpp from SLAB's src\rplugin\ directory to your new project directory. Rename it if you wish. Add it to the project.
4. Enter the SLAB include\ directory to **Project | Settings... | Settings For: All Configurations | C/C++ | Preprocessor | Additional include directories**.
5. For SLAB to find your DLL it must be reside in the SLAB bin\rplugin\ directory. For debug DLLs, enter the name of your DLL appended to the SLAB bin\rplugin\ path as follows: under **Project | Settings... | Settings For: Win32 Debug | Link | Preprocessor | Output file name**, enter "<SLAB install dir>/bin/rplugin/r\*d.dll" where '\*' denotes a name of your choosing ('r' indicates the DLL is a "render" plug-in). For release DLLs, under **Project | Settings... | Settings For: Win32 Release | Link | Preprocessor | Output file name**, enter "<SLAB install dir>/bin/rplugin/r\*.dll".

[Top](#)

[SLAB User Manual](#)

Last Updated: May 29, 2003

# Source Code

- [SLABClient](#)
  - [rdiotic](#)
  - [rplugin](#)
  - [SLABX](#)
  - [Installer](#)
- 

## SLABClient

The source code to the SLABClient library can be found in <install dir>\lib\src\SLABC\. The communication between the client and the server occurs in Wintel byte order (not network byte order). The reasoning behind this is (1) we are a Wintel-based lab and therefore we do not want to swap and unswap for each data value, and (2) I would have to modify both the client code and the server code to use network byte order; as they say, if it ain't broke, don't fix it...

Currently, the SLABWire layer is not part of the SLAB User Release. Thus, to compile the SLAB client, you'll need to set `_SLABW_` to 0 in SLABAPI.cpp. This simply omits support for the Windows-specific SLAB registry variables. So as to not overwrite the User Release library, you'll probably want to remove the "..\..\\" from the "Library | Output file name" project setting.

Should you choose to port the client to another platform, it is recommended you have access to Microsoft Visual C++ for the MFC source code. The client software uses MFC for the sockets implementation. Please post to the SLAB mailing list if you take this on. I'll offer what help I can. It would be great to have Linux and Mac support...

[Top](#)

---

## rdiotic

<install dir>\src\rdiotic\ contains the source code for the monotonic and diotic SLAB Render plug-ins. See [starter project](#).

[Top](#)

---

## rplugin

<install dir>\src\rplugin\ contains the SLAB Render plug-in [starter project](#).

[Top](#)

---

## SLABX

<install dir>\src\SLABX\ contains an SRAPI-based [example application](#).

[Top](#)

---

## Installer

<install dir>\src\Installer\ contains the Visual Studio.NET Setup and Deployment project for the SLAB User Release. This project creates the SLAB.msi Windows Installer file. The Installer project can be used to distribute works based on SLAB. See the SLAB Software Usage Agreement for the legal terms governing SLAB-related distribution.

[Top](#)

---

## [SLAB User Manual](#)

Last updated: May 29, 2003

# Example Application

The SLAB User Release includes the SLABX example application which demonstrates how to write and build a standard SLAB application using Microsoft Visual C++ (hereafter MSVC). The SLAB User Release has only been tested with the Microsoft Visual C++ compiler.

SLABX contains two demos, *Trajectory Demo* and *HRTF Database Per Source Demo*. In the *Trajectory Demo*, a white noise sound source is placed in a circular trajectory about the listener's head. In the *HRTF Database Per Source Demo*, one wave file sound source is placed at (0.5m,0.5m,0.0m) and another is placed at (0.5m,-0.5m,0.0m). The user can select between three different HRTF databases for each source.

All instances of the directory "\\SLAB\\" below should be replaced by your SLAB installation directory.

## SLABX MSVC Dialog Application

SLABX is a standard MSVC dialog application. It was created using the following steps under MSVC 6.0:

### New Project

New | Projects | MFC AppWizard (exe)

MFC AppWizard - Step 1: Select dialog based.

MFC AppWizard - Step 2 of 4: Uncheck About box (keep it simple), check Windows Sockets (for client/server support).

MFC AppWizard - Step 3 of 4: No change.

MFC AppWizard - Step 4 of 4: No change. Finish.

SLAB applications need to be linked with the SLAB, Windows Multimedia, and DirectSound libraries. To avoid "DLL hell", *static* linking is recommended.

### Project Settings

C/C++ Tab | Category: Preprocessor | Additional include directories

Settings For: All Configurations

Enter: \\SLAB\\include

Link Tab | Object/library modules:

Settings For: Win32 Debug

Enter: \\SLAB\\lib\\slabhmd.lib \\SLAB\\lib\\slabcmd.lib (*the SLAB Host and Client Multithreaded Debug libraries*)

(*for a Release Build, use slabhmr.lib and slabcmr.lib*)

Link Tab | Object/library modules:

Settings For: All Configurations

Enter: winmm.lib \\apps\\mssdk\\lib\\dsound.lib (*the Windows multimedia and DirectSound libraries*)

(*for dsound.lib, use the pathname for your installation of DirectSound*)

In SLABX, only two files contain SLAB code, SLABXDlg.cpp and SLABXDlg.h. These two files are discussed below. Each has been edited to emphasize the SLAB-specific code.

## SLABX Interface File

```
// SLABXDlg.h : header file
```

**SLABAPI.h is the only SLAB header file necessary.**

```
#include "SLABAPI.h" // SLAB

class CSLABXDlg : public CDialog
{
// Attributes
protected:
```

**All SLAB interaction occurs through one CSLABAPI object. Declare a CSLABAPI\* to reference this object.**

```
    CSLABAPI*    m_pSLAB;

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(CSLABXDlg)
```

**SLAB reports process-time errors via a Windows command message. You specify the message to receive using SetNotify(). The easiest way to create and handle a message ID is to add an invisible dummy button to a dialog using MSVC's ResourceView. Next, create a message handler for the dummy button (ID ID\_SLAB\_NOTIFY):**

```
    afx_msg void OnSlabNotify();
    //}}AFX_MSG
};
```

## SLABX Implementation File

```
// SLABXDlg.cpp : implementation file
```

**Alter these paths to reference your installation directory!**

```
// wave file to play during HRTF per source demo
#define WAVE_FILE "\\SLAB\\wavs\\voice.wav"

// default SLAB Server IP address
#define DEF_SERVER "localhost"

// HRTF databases for HRTF per source demo
#define HRTF_DB    "\\SLAB\\HRTF\\jdm.slh"        // normal
#define HRTF_DB_ME "\\SLAB\\HRTF\\jdm_me.dat"    // HRIRs equal
#define HRTF_DB_IE "\\SLAB\\HRTF\\jdm_ie.dat"    // ITDs equal
```

```
//-----
// DestroyWindow()
```

```
BOOL CSLABXDlg::DestroyWindow()
{
```

**SLAB should be stopped and freed before the program exits. DestroyWindow() is the safest place to perform clean-up related to Windows resources.**

```

    // if SLAB processing, stop
    if( m_pSLAB )
        OnStop();

    return CDialog::DestroyWindow();
}

//-----
// BOTH DEMOS
//-----

//-----
// OnStop()

void CSLABXDlg::OnStop()
{
    // if timer allocated, kill it (trajectory demo)
    if( m_nTimer )
        KillTimer(m_nTimer);

```

**RenderStop() stops the processing started by RenderStart(). LstHRTFFree() frees an HRTF database. SrcFree() frees all allocated sound sources. Deleting the CSLABAPI object frees all resources associated with SLAB. These four steps are the preferred way to exit SLAB. A time-stamped entry can be made to a log file with LogEntry() and LogTime() to indicate when SLAB was exited. The log file's main purpose is to provide a SLAB error record.**

```

    // stop processing and free sources
    m_pSLAB->RenderStop();
    if( m_idHRTF )
        m_pSLAB->LstHRTFFree( m_idHRTF );
    if( m_idHRTFme )
        m_pSLAB->LstHRTFFree( m_idHRTFme );
    if( m_idHRTFie )
        m_pSLAB->LstHRTFFree( m_idHRTFie );
    m_idHRTF = NULL;
    m_idHRTFme = NULL;
    m_idHRTFie = NULL;
    m_pSLAB->SrcFree();
    m_idSrc0 = NULL;
    m_idSrc1 = NULL;
    m_pSLAB->LogEntry( "SLABX Exit" );
    m_pSLAB->LogTime();

    delete m_pSLAB;
    m_pSLAB = NULL;
}

//-----
// OnSLABNotify() - SLAB error reporting

void CSLABXDlg::OnSlabNotify()
{
    static char strFormat[] = "%s\r\n\r\nDetails:\r\n"
                              "-----"
                              "-----\r\n%s";

    char* strError;

```

```
// if there was a SLAB error, display error, close SLAB
if( m_pSLAB && m_pSLAB->ErrorState() )
{
```

**A one sentence error description is available from ErrorString(). Detailed error information can be obtained from ErrorStack(). Once the error has been reported, the user clears the error with ErrorClear(). The safest response to a SLAB error is to exit SLAB.**

```
    strError = (char*) malloc( strlen( strFormat ) +
    strlen( m_pSLAB->ErrorString() ) +
    strlen( m_pSLAB->ErrorStack() ) );
    sprintf( strError, strFormat, m_pSLAB->ErrorString(),
    m_pSLAB->ErrorStack() );

    m_pSLAB->ErrorClear();
    OnStop();
    AfxMessageBox( strError );
    free( strError );
}
}

//-----
//  TRAJECTORY DEMO
//-----

//-----
//  OnStartTraj()

void CSLABXDlg::OnStartTraj()
{
```

**Users can run SLAB locally on their workstations or remotely on a SLAB Server. Local usage is referred to as "Host Mode" and remote usage is referred to as "Client/Server Mode." The server mode is determined by the type of SLAB object allocated. To use Host Mode, allocate a CSLABHost object using the SLABAPIHost() function. To use Client/Server Mode, allocate a CSLABClient object using the CSLABAPIClient() function. The objects should be accessed through the CSLABAPI interface. When using Client/Server Mode, you need to specify the IP address of the SLAB Server.**

```
// create host-mode or client/server-mode CSLABAPI object
if( m_buttonHostMode.GetCheck() )
{
    m_pSLAB = SLABAPIHost();
}
else
{
    CString    cstrServer;

    // get the SLAB Server IP address from the edit control
    m_editServer.GetWindowText( cstrServer );

    // SLAB Server
    m_pSLAB = SLABAPIClient( (char*) (LPCSTR) cstrServer );
}

// make sure CSLABAPI object allocated
if( !m_pSLAB )
{
    AfxMessageBox( "Failed to allocate SLAB object!" );
}
```

```

    return;
}

```

Use `SetNotify()` to select the window to receive SLAB process-time error messages. You must also specify the command message ID that is sent when an error occurs. SLAB process-time error handling is performed by handling this message. Since SLAB is a real-time system, errors can occur between function calls (e.g. a sound output buffer drop-out). Messaging is a convenient way to catch these errors.

```

// set SLABAPI notification window
// Note: ID_SLAB_NOTIFY is the ID of an invisible dummy button created
// expressly for SLAB notification
m_pSLAB->SetNotify( this, ID_SLAB_NOTIFY );

```

The SLAB log file is primarily used as an error record. This feature is not mandatory. The file to use as the log file is set with `LogFile()` and log file entries are made with `LogEntry()`. Once a log file is specified, SLAB will add entries to the log file whenever an error occurs.

```

// set SLABAPI host log file (ignored in client mode)
m_pSLAB->LogName( "SLABX" );
m_pSLAB->LogEntry( "SLABX Init" );

```

SLAB uses DirectSound for low-latency audio playback. When the write buffer is full, the buffer contents are transferred to the DirectSound output buffer. These parameters are modified to tune the latency/robustness tradeoff. The lower the output latency, the less robust the playback and vice versa. The output latency is basically the output buffer size (in bytes) divided by 176400 bytes/second (2 (stereo) \* 2 bytes/sample (16-bit samples) \* 44100 samples/second (sampling rate)). Thus, the output latency with the configuration below would be 8192 bytes / 176400 bytes/second = 46ms. For WDM drivers (Win2k), an additional 20ms of latency exists due to the WDM KMixer.

```

// init DirectSound output device (output and write buffer sizes in bytes)
m_pSLAB->OutDS( 8192, 512 );

```

`SmoothTime()` selects the time constant used for DSP parameter tracking. As the scene is updated, new DSP parameters are generated. To avoid audible artifacts, the DSP parameters are smoothed from one set of parameters to the next. Higher time constants reduce the chance of audible artifacts, but the system response to scene updates may become sluggish. Lower time constants increase the likelihood of audible artifacts, but provide better responsiveness.

`FIRPoints()` adjusts the size of the FIR filter used to render HRTFs. The FIR filter size can be any power of two between 16 and 128. The size of the filter can be reduced to free up CPU resources.

```

// init signal processing parameters
m_pSLAB->SmoothTime( 15.0 );
m_pSLAB->FIRPoints( 128 );

```

Several functions modify source parameters. `SrcRadius()` sets the radius of the source and affects how fast the sound's volume decreases as the source-listener range increases. `SrcGenNoise()` allocates a noise generator, returning a source ID. The source ID is used as a parameter to the source scene functions to select the source to modify. The amplitude parameter adjusts the minimum and maximum value of the samples generated. This value can be thought of as an *initial* volume control. Once processing is begun, only `SrcGain()` can be used to adjust the volume level.

Sources can be placed using Cartesian coordinates with `SrcLocate()` or using polar coordinates with `SrcLocatePolar()`. The source gain is set with `SrcGain()` using a double valued dB scale (e.g. 0.0 dB = passthru, -96.0 dB = off for 16-bit integer input). `SrcEnable()` enables the rendering of the source. When a source is disabled, rendering calculations are not performed, reducing CPU usage. `SrcEnable( ?, false )` is an efficient way to mute a source (versus the `SrcGain()` method used in this example).

```
// init general source parameters
m_pSLAB->SrcRadius( 0.2 );

// init source-specific parameters
m_dAz = 0.0;
m_idSrc0 = m_pSLAB->SrcGenNoise( 4096 );
m_pSLAB->SrcLocatePolar( m_idSrc0, m_dAz, 0.0, 2.0 );
m_pSLAB->SrcGain( m_idSrc0, 0.0 );
m_pSLAB->SrcEnable( m_idSrc0, true );
```

Three functions modify listener parameters. If the listener position data comes from a head tracker (e.g. Polhemus Fastrak), the function `LstSensorOffset()` should be used to specify the location of the head tracker sensor. `LstPosition()` specifies the position of the listener. `LstHRTF()` specifies the HRTF database used for spatial rendering. The HRTF database is loaded into memory with `LstHRTFLoad()`. To decrease the likelihood of sound output underflow, load HRTFs before rendering is started. Be sure to free the database with `LstHRTFFree()`.

```
// init listener parameters
m_pSLAB->LstSensorOffset( 0.0, 0.0, 0.0 );
m_pSLAB->LstPosition( 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 );

m_idHRTF = m_pSLAB->LstHRTFLoad( HRTF_DB );
if( !m_idHRTF )
{
    AfxMessageBox( "Failed to load HRTF:\r\n" HRTF_DB );
    OnStop();
    return;
}

m_pSLAB->LstHRTF( m_idHRTF );
```

**RenderStart()** initiates rendering.

```
// start rendering
m_pSLAB->RenderStart( RENDER_SPATIAL );

// start a Windows timer for updating trajectory
// note: 55ms is the shortest period possible with SetTimer()
m_nTimer = SetTimer( 1, 55, NULL );
}

//-----
// OnTimer()

void CSLABXDlg::OnTimer(UINT nIDEvent)
{
    // increment azimuth
    m_dAz += 0.055; // 1 rad/s
    if( m_dAz >= dPIx2 )
        m_dAz -= dPIx2;
```

**SLAB does not have an update rate per se. Each scene update causes the DSP parameters to be updated almost instantaneously (within 64 samples). Thus, the frequency of scene updates determines the update rate of SLAB. In this example, a scene update occurs every 55ms, yielding an effective update rate of 18Hz. A typical SLAB update rate is 120Hz.**

```
// update source location
m_pSLAB->SrcLocatePolar( 0, m_dAz, 0.0, 2.0 );

CDialog::OnTimer(nIDEvent);
}

//-----
//  HRTF PER SOURCE DEMO
//-----

//-----
//  OnStartHrtf()

void
CSLABXDlg::OnStartHrtf()
{
```

**Most of this function is identical to OnStartTraj() (see above). This function differs only in the way the sources are allocated and placed. SrcFile() is called twice to allocate two looped wave file sound sources. SrcLocate() places the sources using Cartesian coordinates.**

```
// init source-specific parameters
m_idSrc0 = m_pSLAB->SrcFile( WAVE_FILE );
m_idSrc1 = m_pSLAB->SrcFile( WAVE_FILE );
m_pSLAB->SrcLocate( m_idSrc0, 0.5, 0.5, 0.0 );
m_pSLAB->SrcLocate( m_idSrc1, 0.5, -0.5, 0.0 );
m_pSLAB->SrcGain( m_idSrc0, 0.0 );
m_pSLAB->SrcGain( m_idSrc1, 0.0 );
m_pSLAB->SrcEnable( m_idSrc0, true );
m_pSLAB->SrcEnable( m_idSrc1, true );
}

//-----
void CSLABXDlg::OnSrcInorm()
{
    if( !m_pSLAB ) return;
```

**LstHRTF() can be used to change the HRTF database during real-time rendering. This allows the user to compare the sound of one HRTF database to another. In this example, three HRTF databases are available: one normal (the default SLAB HRTF database), one modified so that all HRIRs equal the 0az,0el magnitude, and another modified so that all ITDs equal the 0az,0el ITD (0az,0el ITD = 0 samples). Thus, the two primary spatialization cues are isolated and compared. The functions OnSrc1me() and OnSrc1ie() load the modified HRTF databases (see SLABXDlg.cpp).**

**SLAB supports listening to each sound source through its own HRTF database. If a second parameter to LstHRTF() exists, it specifies a sound source/HRTF database association.**

```
// normal map, source 0
m_pSLAB->LstHRTF( m_idHRTF, m_idSrc0 );
```

```

}

//-----
void CSLABXDlg::OnSrc1mute()
{
    if( !m_pSLAB )    return;

```

**SrcGain()** allows sources to be gained or muted during real-time rendering. In this function, a gain of -120.0 dB is chosen as an arbitrary mute value. Since -120.0 dB is below the dynamic range threshold for 16-bit integer, it will completely attenuate the source. Note: even if a source is not heard, it is still being rendered and, thus, consuming computational resources!

```

    if( m_bMute1 )
    {
        // mute gain, -120dB
        // (note: 20log10(1/2^16) = -96dB)
        m_pSLAB->SrcGain( m_idSrc0, -120.0 );
        m_buttonMute1.SetWindowText( "Unmute" );
    }
    else
    {
        // pass-thru gain, 0dB
        m_pSLAB->SrcGain( m_idSrc0, 0.0 );
        m_buttonMute1.SetWindowText( "Mute" );
    }

    m_bMute1 = !m_bMute1;
}

```

```

//-----
void CSLABXDlg::OnSrc2norm()
{
    if( !m_pSLAB )    return;

```

**This function is similar to OnSrc1norm(), but sets the HRTF database used for the second sound source.**

```

    // normal map, source 1
    m_pSLAB->LstHRTF( m_idHRTF, m_idSrc1 );
}

```

```

//-----
void CSLABXDlg::OnSrc2mute()
{
    if( !m_pSLAB )    return;

```

**This function is similar to OnSrc1mute(), but adjusts the gain for the second sound source.**

```

    if( m_bMute2 )
    {
        // mute gain, -120dB
        // (note: 20log10(1/2^16) = -96dB)
        m_pSLAB->SrcGain( m_idSrc1, -120.0 );
        m_buttonMute2.SetWindowText( "Unmute" );
    }
    else
    {
        // pass-thru gain, 0dB
        m_pSLAB->SrcGain( m_idSrc1, 0.0 );
        m_buttonMute2.SetWindowText( "Mute" );
    }

```

```
    }  
    m_bMute2 = !m_bMute2;  
}
```

## [SLAB User Manual](#)

Last Updated: May 29, 2003

# Applications and Utilities

- [SLABscape](#)  
SLABscape provides a graphical user interface for experimenting with the SLAB Render API scene parameters. It is also used to alter the SLAB Registry Settings (e.g., Default HRTF).
- [SLABServer](#)  
The SLABServer application allows a workstation to be dedicated as a stand-alone SLAB virtual audio server.
- [slabtools](#)  
The slabtools are MATLAB utilities for visualizing and manipulating HRTF data.
- [Example Application](#)  
The SLABX example application demonstrates the use of the SLAB Render API.

---

## [SLAB User Manual](#)

Last updated: June 30, 2003

# SLABscape

SLABscape provides a graphical user interface to the SLAB Render API. SLABscape also allows the user to specify sound source trajectories, enable Fastrak head tracking, edit and play SLAB Scripts, A/B different rendering strategies, and visualize the environment via a Direct3D display.

Contents:

- [Menus](#)
  - [Toolbar](#)
  - [Shortcuts](#)
  - [Source Dialog](#)
  - [Environment Dialog](#)
  - [Listener Dialog](#)
  - [Fastrak Settings Dialog](#)
  - [SLAB Settings Dialog](#)
  - [SLABScript Editor](#)
  - [3D View](#)
  - [Status Bar](#)
- 

## Menus

Menu definitions:

- **File**
  - **New** - initializes scene parameters to their default settings.
  - **Open...** - prompts to open a **.scn** "SLAB Scene" file.

Note: SLABscape does not strictly adhere to the MFC (Microsoft Foundation Classes) view/document model. If you attempt to reopen an open scene, MFC prevents the open and does nothing. As a work-around, simply click the File New button before reopening.

  - **Save** - saves the current scene. The sound source and listener controlled by controller (tracker or Firefly) setting is not saved, nor is the Render Plug-In ID if it is above RENDER\_PLUGIN. This is because it is not known if the controller or renderer will be available next time SLABscape is executed.
  - **Save As...** - prompts to save the current scene to a file.
  - **Most Recently Used Files** - lists recently used scene files.
  - **Exit** - exits the application.
- **View**
  - **Scene**

- **Source Dialog...** - displays the [Source Dialog](#).
- **Environment Dialog...** - displays the [Environment Dialog](#).
- **Listener Dialog...** - displays the [Listener Dialog](#).
- **Controllers**
  - **Fastrak Settings...** - displays options for configuring a Polhemus Fastrak tracker.
  - **Firefly Settings...** - displays options for initializing and configuring Firefly. These controls are currently implemented in-house only.
- **Simulation Settings...** - displays simulation parameter options for tick update rate and hand control.
- **Sources Displayed** - sets the number of sources displayed in the visual display.
- **Parameter Overlay** - overlays the 2D view with source and listener scene information. All distances are in meters, all angles in radians. Azimuth, elevation, and range are relative to the origin.
  - Sources: x, y, z, azimuth, elevation, range, gain
  - Listener: x, y, z, yaw, pitch, roll
- **Update Rate** - displays the update rate of the SLAB Renderer and the visual display (the value in parentheses) on the status bar.
- **Monitor SLAB Autos** - if SLAB is in host mode, monitors the positions of the sources and the listener as defined by the SLAB Renderer (the SLAB Renderer can automatically update its own parameters when performing a trajectory or when scripting).
- **Toolbar** - controls the display of the toolbar.
- **Status Bar** - controls the display of the status bar.
- **SLAB**
  - **Render** - starts or stops the SLAB Renderer. Renders using the currently selected renderer (see Plug-Ins menu).
  - **SLAB Settings...** - provides access to all pre-render-time SLAB parameters (except materials, see [Environment Dialog](#)). The signal processing parameters, script settings, network settings, and output settings are set using this dialog.
  - **SLAB Registry Settings...** - lists the SLAB installation registry settings. The user can modify the default HRTF and the default HRTF directory registry settings using this dialog.
  - **SLABScript Editor...** - displays the [SLABScript Editor](#).
- **Plug-Ins**
  - **About Plug-Ins...** - displays information about all plug-ins found in the SLAB Render plug-in directory (<install dir>\bin\rplugin). The SLAB Renderer must be started before plug-in information is available.
  - **Installed Plug-Ins** - all SLAB Render plug-ins will be added to the Plug-Ins menu after the SLAB Renderer is started.

Warning: Only the Left-Monotic, Right-Monotic, Diotic, and DioticV plug-ins are

affected by the M/D (monotic/diotic) gain setting on the source dialog. All other plug-ins use the source gain setting. Thus, plug-ins that do not attenuate the sound signal in their rendering model can result in a significant volume increase when selected! (e.g., "Spatial" attenuates the signal due to spherical spreading and the head-related impulse response in addition to the gain setting; "Plug-In Example" does not attenuate beyond the gain setting.)

The following plug-ins are included in the SLAB User Release:

- **Spatial** - selects the spatial SLAB Render plug-in.
  - **Left-Monotic** - selects the left-monotic SLAB Render plug-in.
  - **Right-Monotic** - selects the right-monotic SLAB Render plug-in.
  - **Diotic** - selects the diotic SLAB Render plug-in.
  - **DioticV** - selects the variable diotic (source  $y = 0.0$ ) / monotic (source  $y$  left or right) SLAB render plug-in.
  - **Plug-In Example** - selects the example SLAB render plug-in (diotic).
- **Help**
    - **Help...** - displays this help page.
    - **About...** - displays about dialog box.

[Top](#)

---

## Toolbar

Toolbar button definitions (from left to right):

- **New** - see [Menus](#).
- **Open...** - see [Menus](#).
- **Save** - see [Menus](#).
- **Source Dialog...** - see [Menus](#).
- **Environment Dialog...** - see [Menus](#).
- **Listener Dialog...** - see [Menus](#).
- **SLAB Render** - see [Menus](#).
- **SLAB Settings...** - see [Menus](#).
- **HRTF 1** - selects HRTF 1 defined in [Listener Dialog](#).
- **HRTF 2** - selects HRTF 2 defined in [Listener Dialog](#).
- **HRTF 3** - selects HRTF 3 defined in [Listener Dialog](#).
- **Spatial** - see [Menus](#).
- **Left Monotic** - see [Menus](#).
- **Right Monotic** - see [Menus](#).

- **Diotic** - see [Menus](#).
- **Parameter Overlay** - see [Menus](#).
- **Zoom Out** - zooms-out on the 2D view. The zoom level changes the source and listener placement slider ranges.
- **Zoom In** - zooms-in on the 2D view. The zoom level changes the source and listener placement slider ranges.
- **3D View** - toggles between the 2D view and the [3D View](#). The 3D View is rendered using Direct3D, so the quality of the display is determined by the Direct3D features of your graphics card.

[Top](#)

---

## Shortcuts

Command	Shortcut Key
<b>Sound Sources</b>	
Control source number x	1, 2, 3, 4
Cycle control	c
Reset source location forward	End
Up	u
Down	d
In	i
Out	o
Mute/UnMute	Ins
Increase gain	g
Decrease gain	a
<b>Sound Source Trajectory</b>	
Slower	s
Faster	f
Start/Stop	Space
Change direction	Enter
Step trajectory	.
<b>Listener</b>	
Move to origin	Home
Forward	Up Arrow
Backward	Down Arrow

Left	Left Arrow
Right	Right Arrow
Up	Page Up
Down	Page Down
Spin	/

[Top](#)

---

## Source Dialog

The Source Dialog allows the user to manipulate source parameters while rendering. The dialog's layout is similar to an audio mixer with a channel strip for each sound source. These strips contain the familiar gain fader and mute button, but also include a source motion droplist, a relative source location dialog button (REL), a current source controlled button (channel #), a pause/play button (PAU), and a rewind button (REW).

The source motion droplist contains five options: None, Circle, Rectangle, Track, and Firefly. The Circle and Rectangle trajectories are generated by SLABScape, *not* the SLAB Renderer (the SLAB Renderer has a built in trajectory mechanism that is currently only supported by the [SLABScript Editor](#)). The Circle and Rectangle trajectories occur in a horizontal plane. The Circle radius is defined by the initial radius. The Rectangle dimensions are determined by the initial location of the source. Keyboard [shortcuts](#) provide further trajectory control. The Track option enables source tracking. This causes the sound source to follow the location of a Polhemus Fastrak tracker receiver. The tracker is opened using the "Open Tracker" button on the "Tracker Settings" dialog (View | Controllers menu). Firefly allows a source to be located using USB camera capture. This option is currently implemented in-house only; it does nothing in the User Release.

The REL button pops-up the Relative Source Location dialog. This allows the placement of a source relative to the listener's head using azimuth, elevation, and range coordinates. This operation places the listener at the origin. Relative source placement allows one to listen to a specific location in an HRTF database or to investigate the effects of HRTF interpolation.

The channel# button determines the "current source." The current source is controlled via the Location sliders on the right side of the dialog and by using keyboard [shortcuts](#).

The Monotic-Diotic Gain slider sets the source gain for the Left-Monotic, Right-Monotic, Diotic, and DioticV renderers. Warning: If both release and debug versions of the plug-in exist, the debug version is not affected by this slider. Thus, a signification increase in volume may result if the debug version is chosen!

The "Allocate Sources..." button pops-up the "Source Allocation" dialog for allocating sound sources (Noise, Sine, Square, and File).

The "Source Radius" droplist sets the radius of the current sound source. The "Source Spread" droplist sets the spherical spreading coefficient for the current sound source (0 = none, 1 = normal, 2 = exaggerated).

[Top](#)

---

## Environment Dialog

The Environment Dialog controls features of the listening environment. These features include the dimensions of a rectangular room, the materials of the room surfaces, and the number of sound images modeled. The number of sound images is controlled by the Reflections array of checkboxes. D refers to the direct path, XP to the positive x-axis plane, and so on. The surface materials are selected via the Surface Materials dialog displayed by pressing the Materials... button.

[Top](#)

---

## Listener Dialog

The Listener Dialog controls the listener location and orientation, tracking options, and HRTF database selection. The Tracker Enable button enables head tracking. This causes the listener to follow the location of a Polhemus Fastrak tracker receiver. The tracker is opened using the "Open Tracker" button on the "Tracker Settings" dialog (View | Controllers menu). Selecting the Orientation Only button locks the listener at the origin while listener yaw, pitch, roll continue to be updated.

The HRTFs... button displays the "HRTF Databases" dialog. This dialog sets the HRTF databases loaded by the HRTF 1, HRTF 2, and HRTF 3 toolbar buttons.

[Top](#)

---

## Fastrak Settings Dialog

These settings initialize a Polhemus Fastrak electromagnetic tracking peripheral.

- **Com Port** - Select the com port connected to the Fastrak. The Fastrak communication settings (DIP switches) must be set to 115200 bps, 8 data bits, no parity, no flow control.
- **Hemisphere** - Direction of valid Fastrak transmitter hemisphere (see Fastrak manual).
- **Receiver 2** - Selecting "Hand" will enable and map the second Fastrak receiver to the Hand. (The Hand is currently in development.) Enabling a second receiver drops the Fastrak update rate from 120Hz to 60Hz.
- **System Com** - Typically, SLABScape sets the com settings using the Microsoft Comm API. Some com devices do not support this feature (e.g., some USB-to-serial adapters) and use the Device Manager com settings instead. These devices will cause "Open Tracker" to fail. Selecting this option bypasses the Comm API settings.

Note: for com devices that support Comm API com settings, this option will cause the com port to be initialized with the most recent Comm API settings. Thus, only use this option if you have to.

- **Open Fastrak** - Initiates communication with the Fastrak. Opening the Fastrak sets the

current settings and cannot be canceled.

- **Save Settings** - Saves the current dialog settings in the registry.
- **Restore Defaults** - Restores dialog settings default settings.
- **OK** - Sets dialog settings.
- **Cancel** - Resets dialog settings to previous settings.

[Top](#)

---

## SLAB Settings Dialog

These settings initialize SLAB prior to rendering and are described in more detail as non-process-time functions in the SRAPI Reference Manual.

- **Signal Processing Parameters** - see the SRAPI Reference Manual.
- **Script Settings** - if the "Process script file while rendering" checkbox is checked, the specified file will be parsed the next time the SLAB Renderer is started.
- **Network Settings** - If the "Run SLAB in Host Mode" checkbox is not checked, SLABScope attempts to connect to the specified SLAB Server. SLABScope uses the slabserv.ini file in SLAB's bin directory to initialize the droplist.
- **Output Settings** - for an explanation of the DirectSound Settings see the SRAPI Reference Manual. If the "Output to memory and write to file" checkbox is checked, SLABScope routes sound output to the specified file. In this mode, no sound is heard through the sound peripheral. Since rendering does not occur in real-time, the SLABScope user interface is not synchronized to sound output. For simple scenes, SLAB will render faster than real-time; for complex scenes, SLAB will render slower than real-time. This option is primarily for debugging, but can be used with the SLABScript Editor to render a "canned" dynamic scene. Output is stopped when the SLAB Renderer is stopped.

Caution: Sound file output is generated at 172 kb/s, so do not forget to stop the SLAB Renderer!

[Top](#)

---

## SLABScript Editor

The SLABScript Editor allows for the creation and auditioning of SLAB scripts.

Button definitions:

- **Add Event** - displays a dialog for adding SLABScript commands. The default parameters are all set to zero. Be sure to change this value for Source Index (valid values = 1-4).
- **Open...** - opens a previously saved script.
- **Save As...** - saves the current script.
- **PrepScope** - prepares SLABScope for script playback:
  - enables script processing (enables checkbox: SLAB Settings | Script Settings |

Process script file while rendering)

- sets the SLAB Settings script file to the last script loaded or saved (sets edit control: SLAB Settings | Script Settings | File)
- enables SLAB auto monitoring (enables menu item: View | Monitor SLAB Autos)

To audition a script, press the PrepScape button and then start the Spatial renderer (select Spatial Render plug-in, press SLAB Render toolbar button).

[Top](#)

---

## 3D View

The 3D View displays a Direct3D view of the SLABscape environment. Right-clicking on the 3D View brings up a context-sensitive menu for adjusting various 3D View options. If your graphics adapter or desktop display settings do not support Direct3D acceleration, this button will be disabled.

[Top](#)

---

## Status Bar

The two status indicator panes in the lower-right of the SLABscape window show the following:

- Update Rate Indicator - the number without parentheses indicates the rate at which the SLAB Renderer is updated; the number within parentheses indicates the update rate of the visual display.
- Clip and Underflow Indicator - the "C=" field indicates the number of renderer mixer clips (implementation renderer-dependent); the "U=" field indicates the number of DirectSound underflows.

[Top](#)

---

## [SLAB User Manual](#)

Last Updated: October 1, 2003

# SLABServer

The SLABServer application allows a workstation to be dedicated as a stand-alone SLAB server. In this configuration, the entire computational load is transferred to the server. This allows for more robust rendering and frees user workstation resources. To communicate with the SLAB server, allocate a CSLABClient object using SLABAPIClient().

## The SLABServer User Interface

- **Server Log** - displays SRAPI commands as they arrive. Since the following functions can be received quite frequently, they are not logged: SrcLocate(), SrcLocatePolar(), SrcGain(), and LstPosition().
- **Monitor** - monitors the following parameters at a 15Hz update rate:
  - listener position (meters and degrees)
  - sound source location (meters)
  - listener-relative sound source azimuth and elevation (degrees)
  - sound source gain (linear gain scalar, not dB)
  - number of mixer clips
  - number of DirectSound underflows
  - number of packets received
  - number of packets received per second (updated every second)
- **Listen for Connection** - begins listening for a socket connection from a SLAB client.
- **Test** - spatially renders a noise sound source to the left and a sine sound source to the right of the listener. Note: since the sounds are *spatially* rendered, each will be heard in both output channels
  - listener: ( 0.1m, 0.2m, 0.3m, 1.0deg, 2.0deg, 3.0deg )
  - sound source 1: ( 0.1m, 1.2m, 0.3m ), gain = -2.0dB
  - sound source 2: ( 0.2m, -1.2m, -0.3m ), gain = -1.0dB
- **Turn Monitor On** - enables the Monitor.
- **Exit** - closes SLABServer.

---

[SLAB User Manual](#)

Last Updated: December 19, 2002

# slabtools

The *slabtools* are MATLAB utilities for visualizing and manipulating HRTF data and for visualizing SLAB processing. To use the slabtools, the slabtools directory should be added to your MATLAB path. General slabtools help can be obtained by typing "help slabtools" at the MATLAB command prompt. Individual tool help is available by typing "help <toolname>".

## Definition of the variables *map*, *hrir*, and *itd*

These variables are similar to the Snapshot variables of the same name. Snapshot is an HRTF measurement system developed by Crystal River Engineering.

- *hrir* contains HRIR taps stored in rows. All left ear HRIRs are followed by all right ear HRIRs (e.g. the right ear response `hrir(:, i+size(map, 2))` corresponds to the left ear response `hrir(:, i)`).
- *itd* contains the interaural time delay (in samples).
- *map* maps azimuth and elevation database locations to *hrir* and *itd* indices (see *hindex*). Each azimuth and elevation pair is a two-element column vector `[ el; az ]`.

The number of columns in *map*, *hrir*, and *itd* is HRTF database-dependent. Since *hrir* contains the left and right ear responses, it will have twice the number of columns as *map* and *itd*.

The use of *map*, *hrir*, and *itd* is identical to Snapshot with the following exception:

- SLAB's *map* variable is grouped by azimuth (e.g. `[ [ 90; 180], [72; 180] ]`); Snapshot's, by elevation (e.g. `[ [ 90; 180], [90; 150] ]`)

## sarc

sarc is the SLAB HRTF archive format. The goal of the sarc format is to provide a flexible MATLAB HRTF data structure and storage format for HRTF manipulation, visualization, and analysis. To learn more about sarc, type "help sarc" at the MATLAB command prompt.

- **sarc is presently in development!** We are currently upgrading our HRTF measurement system to the AuSIM HeadZap system. Thus, the sarc format will change to accommodate the data generated by HeadZap. I would like to support HRTF data from other formats as well (e.g., CIPIC). Any suggestions regarding the sarc format or similar efforts welcome! Please send suggestions to Joel D. Miller at [jdmiller@mail.arc.nasa.gov](mailto:jdmiller@mail.arc.nasa.gov).

## Examples

SLAB\HRTF\jdm\_ie.slh was created as follows:

```
>> mm itd( 'jdm.slh', 'jdm_ie.slh', 0, 0 );
```

SLAB\HRTF\jdm\_he.slh was created as follows:

```
>> mm hrir( 'jdm.slh', 'jdm_he.slh', 0, 0 )
```

## map2map()

map2map() is the intellectual property of Jonathan Abel and is covered under the same usage

restrictions outlined in the SLAB Software Usage Agreement. Many thanks to Jonathan Abel for allowing map2map() to be released with the SLAB User Release!

---

[SLAB User Manual](#)

Last Updated: December 19, 2002

# Appendix

- [Glossary](#)
  - [Known Issues](#)
- 

## Glossary

Client/Server Mode	Client/Server Mode refers to running SLAB remotely on a separate workstation.
Diotic	Diotic is a Render plug-in that supports diotic auditory display.
DioticV	DioticV is a Render plug-in that supports diotic, left-monotic, and right-monotic auditory display.
DSP Function	A DSP Function is a C++ class which is "plugged-into" SLAB's signal flow architecture. This occurs in SLAB's SLABWire layer and is hidden from the SLAB Render API user.
Host Mode	Host Mode refers to running SLAB locally on the same workstation as the user's application.
HRTF Database	Set of HRIRs and ITDs measured about a sphere specified by azimuth and elevation.
HRTF Map	See HRTF Database.
LeftMonotic	LeftMonotic is a Render plug-in that supports left-monotic auditory display.
MSVC++	Microsoft Visual C++. The SLAB development environment.
RightMonotic	RightMonotic is a Render plug-in that supports right-monotic auditory display.
Render plug-in	Render plug-ins are DLLs that can be loaded and inserted in the SLAB rendering architecture. The SLAB Render API supports five Render plug-ins: Spatial, DioticV, Diotic, LeftMonotic, and RightMonotic. A Render plug-in is inserted into SLABWire as a "DSP Function."
SLAB	SLAB is short for Sound Lab. What "SLAB" refers to is context-dependent. Generally speaking, SLAB refers to the entire collection of SLAB-related material, applications, libraries, documentation, etc. Frequently, however, "SLAB" will refer to one of the constituent parts (e.g. the SLAB Spatial Renderer, the SLAB Server, the SLABAPI Libraries, etc.).
SLAB Client	A user application that uses the SLAB Client API to communicate with the SLAB Server.
SLAB Client Library	The SLAB Client Library contains the SLAB Render Client API.

SLAB Host Library	The SLAB Host Library contains the SLAB Render Host API.
SLAB Libraries	There are four SLAB libraries: Wire, Host, Client, and Server.
SLABWire Library	The SLABWire Library contains the SLABWire layer, a "digital wire" or, more specifically, a sample stream input to DSP to sample stream output chain.
SLAB Render API	The SLAB Render API (application programming interface) (SRAPI) is the interface through which the user interacts with a SLAB Render plug-in. This API is used to specify scene parameters and to control rendering. There are two SLAB Render APIs of interest to the SLAB user: the Host API (Host Mode) and the Client API (Client/Server Mode). A third API exists for writing server applications, the Server API.
SLAB Render API Libraries	A subset of the SLAB Libraries referring to the Host, Client, and Server libraries.
SLABWire Layer	This layer is contained in the SLABWire Library and is hidden from the SLAB Render API user. The SLAB Render API is built on top of this layer.
SLAB Server	A SLAB Workstation configured to run in Client/Server Mode. The SLABServer application runs on this workstation.
SLAB Server Library	The SLAB Server Library contains the SLAB Render Server API.
SLAB Workstation	The SLAB Workstation is the workstation on which the SLAB renderer is running.
SLABscape Application	An application that demonstrates many of the features of SLAB.
SLABServer Application	The application that runs on the SLAB Server to provide SLAB rendering services to a SLAB Client.
Spatial	Spatial is a Render plug-in for rendering HRTF-based virtual acoustic environments.
SRAPI	SLAB Render API.
VAE	Virtual Acoustic Environment.

[Top](#)

---

## Known Issues

- SLAB does not detect all occurrences of DirectSound buffer underflow. Usually, underflow results in an underflow counter increment. But, sometimes, stuttering is heard without an underflow increment. This is most likely to occur when the CPU is at

maximum usage or a CPU usage spike occurs (e.g. reading a file). A missed underflow can be caused by a late Windows timer update, allowing the DirectSound pointers to wrap around the buffer to a valid location.

- SLAB's client/server implementation is not robust. The following behavior has been noticed when running SLABscape in client/server mode with SLABServer. If a source is moving in a trajectory, the client packet receive function will most likely time-out at some point. The packets are probably overwhelming the sockets interface. Also, renderer enumeration can fail with SLABscape receiving an incomplete enumeration info packet. The source of this behavior is not known.
- The SLABscape "hand" is in development and not documented.
- slabref.pdf Acrobat Issues: (1) light blue background of SRAPI function name not in pdf, (2) class hierarchy diagrams not in pdf (Acrobat doesn't support PNG), (3) sections in strange order (order automatically generated by Acrobat).
- ScriptRead() and ScriptWrite() are currently limited to 32 sources in client/server mode.
- Under Windows98, there appears to be a somewhat severe memory leak when performing several (100s) of RenderStart()/RenderStop()'s. Unfortunately, it is not reproducible, and, thus, very hard to track down. Under Windows2000, the leak is less severe (or possibly non-existent (v5.1.0 test)). This bug is thought to be due to thread behavior.
- Under Windows98, a case has been noticed where the left and right channels swap when performing several (100s) of RenderStart()/RenderStop()'s. This bug is thought to be related to the bug above. This behavior has not been noticed under Windows2000.

[Top](#)

---

[SLAB User Manual](#)

Last Updated: June 27, 2003